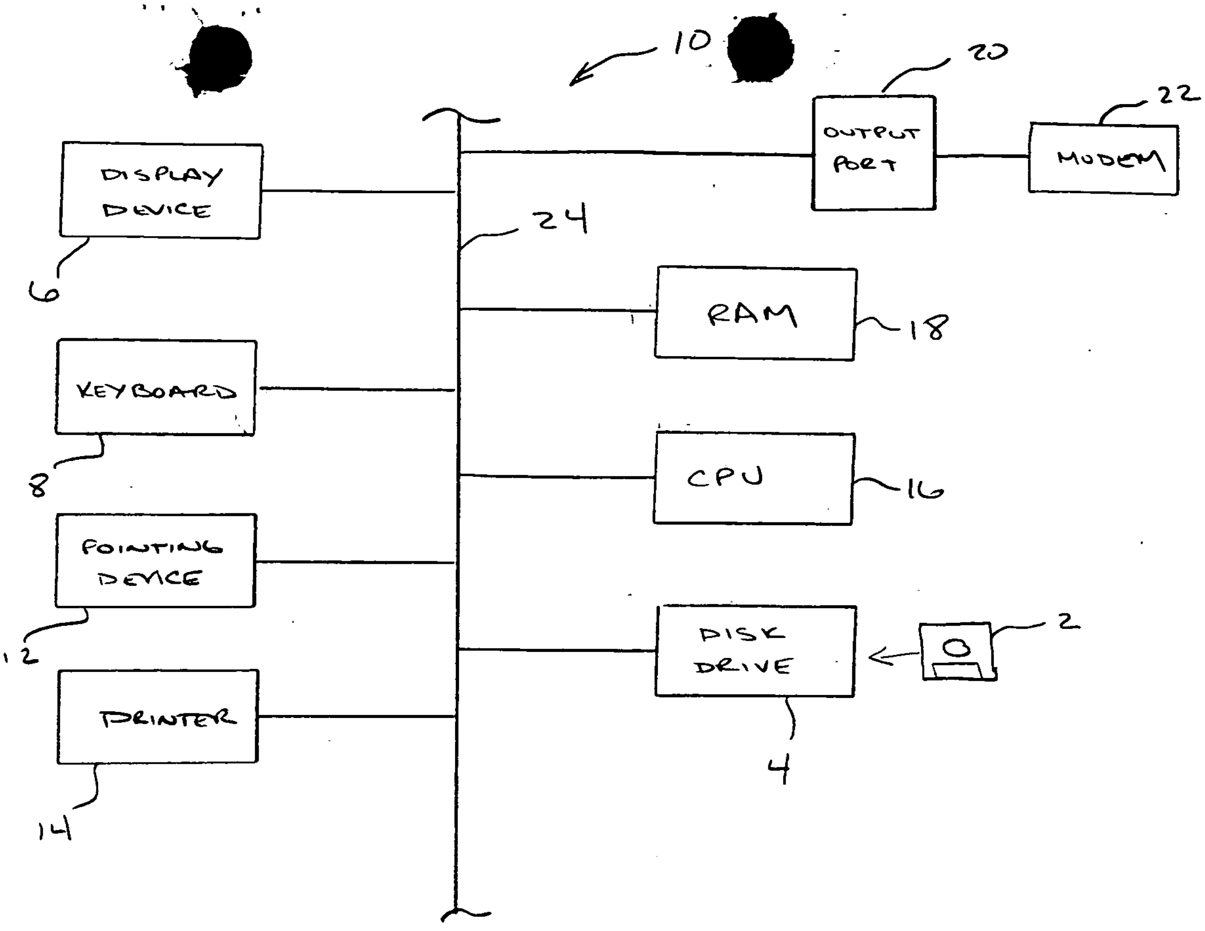

10
OUTPUT PORT
20
MODEM
22
DISPLAY DEVICE
6
24
RAM
18
KEYBOARD
8
CPU
16
POINTING DEVICE
12
DISK DRIVE
2
4
PRINTER
14

FIG. 1

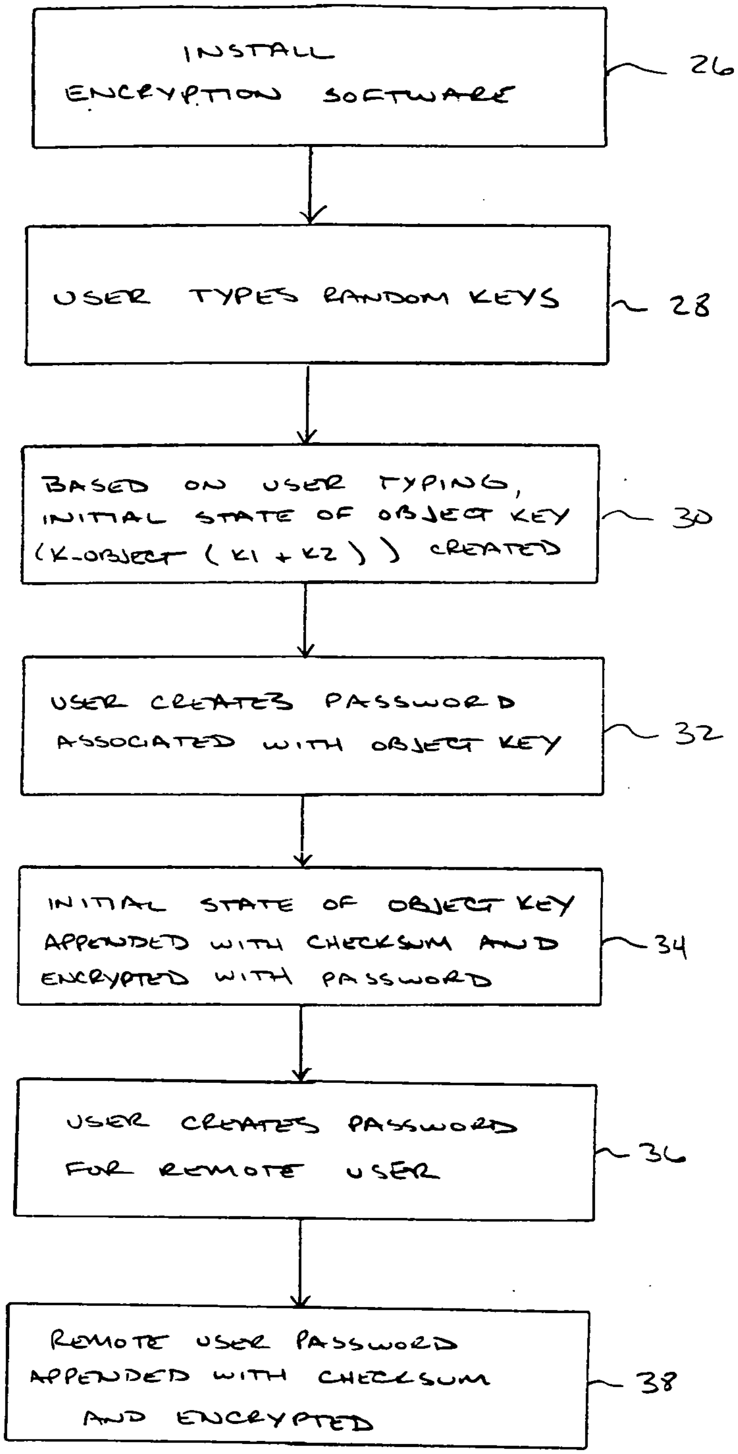
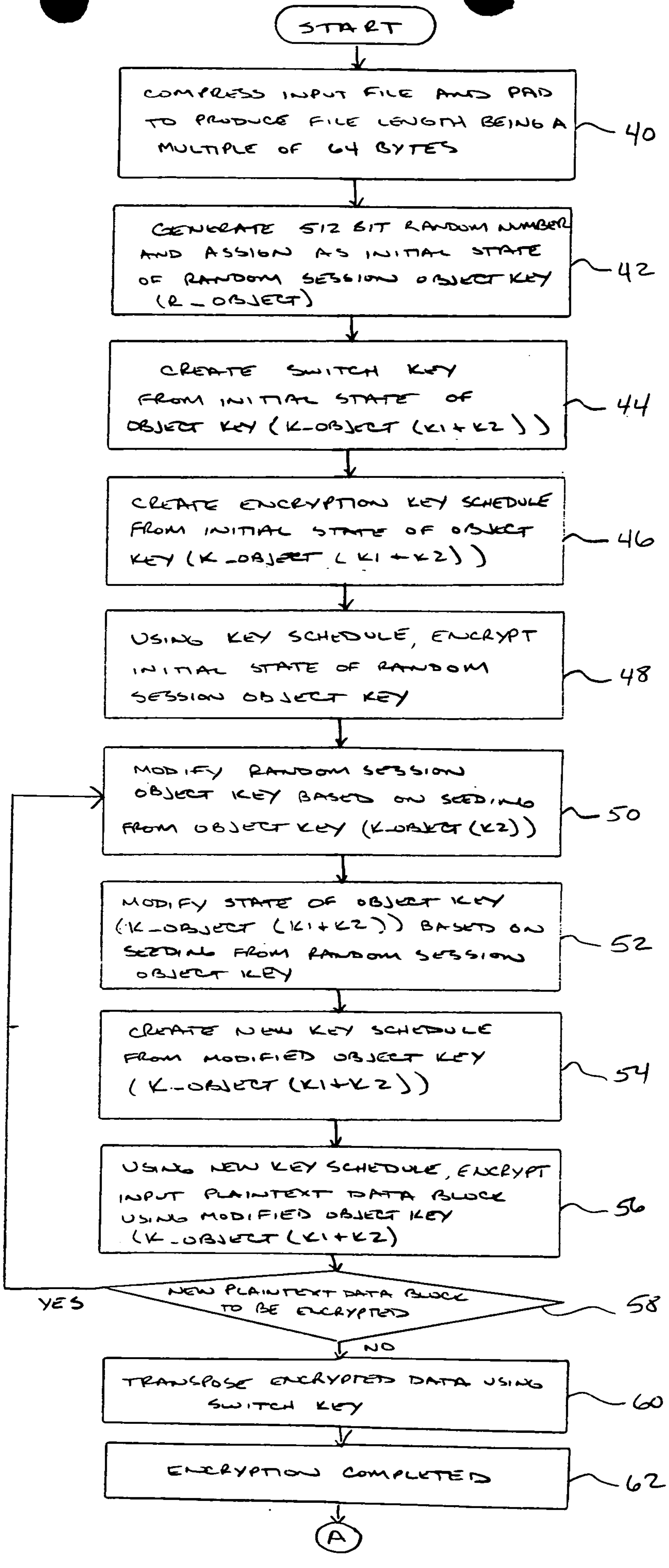

INSTALL ENCRYPTION SOFTWARE
26
USER TYPES RANDOM KEYS
28
BASED ON USER TYPING, INITIAL STATE OF OBJECT KEY (K-OBJECT (K1 + K2)) CREATED
30
USER CREATES PASSWORD ASSOCIATED WITH OBJECT KEY
32
INITIAL STATE OF OBJECT KEY APPENDED WITH CHECKSUM AND ENCRYPTED WITH PASSWORD
34
USER CREATES PASSWORD FOR REMOTE USER
36
REMOTE USER PASSWORD APPENDED WITH CHECKSUM AND ENCRYPTED
38

FIG. 2

FIG. 3


START
COMPRESS INPUT FILE AND PAD TO PRODUCE FILE LENGTH BEING A MULTIPLE OF 64 BYTES
40
GENERATE 512 BIT RANDOM NUMBER AND ASSIGN AS INITIAL STATE OF RANDOM SESSION OBJECT KEY (R_OBJECT)
42
CREATE SWITCH KEY FROM INITIAL STATE OF OBJECT KEY (K_OBJECT (K1+K2))
44
CREATE ENCRYPTION KEY SCHEDULE FROM INITIAL STATE OF OBJECT KEY (K_OBJECT (K1+K2))
46
USING KEY SCHEDULE, ENCRYPT INITIAL STATE OF RANDOM SESSION OBJECT KEY
48
MODIFY RANDOM SESSION OBJECT KEY BASED ON SEEDING FROM OBJECT KEY (K_OBJECT (K2))
50
MODIFY STATE OF OBJECT KEY (K_OBJECT (K1+K2)) BASED ON SEEDING FROM RANDOM SESSION OBJECT KEY
52
CREATE NEW KEY SCHEDULE FROM MODIFIED OBJECT KEY (K_OBJECT (K1+K2))
54
USING NEW KEY SCHEDULE, ENCRYPT INPUT PLAINTEXT DATA BLOCK USING MODIFIED OBJECT KEY (K_OBJECT (K1+K2)
56
NEW PLAINTEXT DATA BLOCK TO BE ENCRYPTED
58
YES
NO
TRANSPOSE ENCRYPTED DATA USING SWITCH KEY
60
ENCRYPTION COMPLETED
62
A

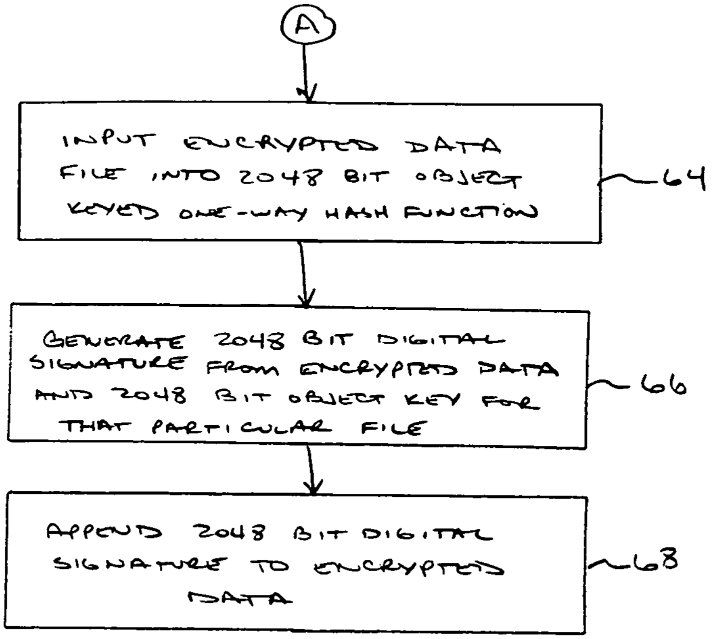

A
INPUT ENCRYPTED DATA FILE INTO 2048 BIT OBJECT KEYED ONE-WAY HASH FUNCTION
64
GENERATE 2048 BIT DIGITAL SIGNATURE FROM ENCRYPTED DATA AND 2048 BIT OBJECT KEY FOR THAT PARTICULAR FILE
66
APPEND 2048 BIT DIGITAL SIGNATURE TO ENCRYPTED DATA
68

FIG. 4

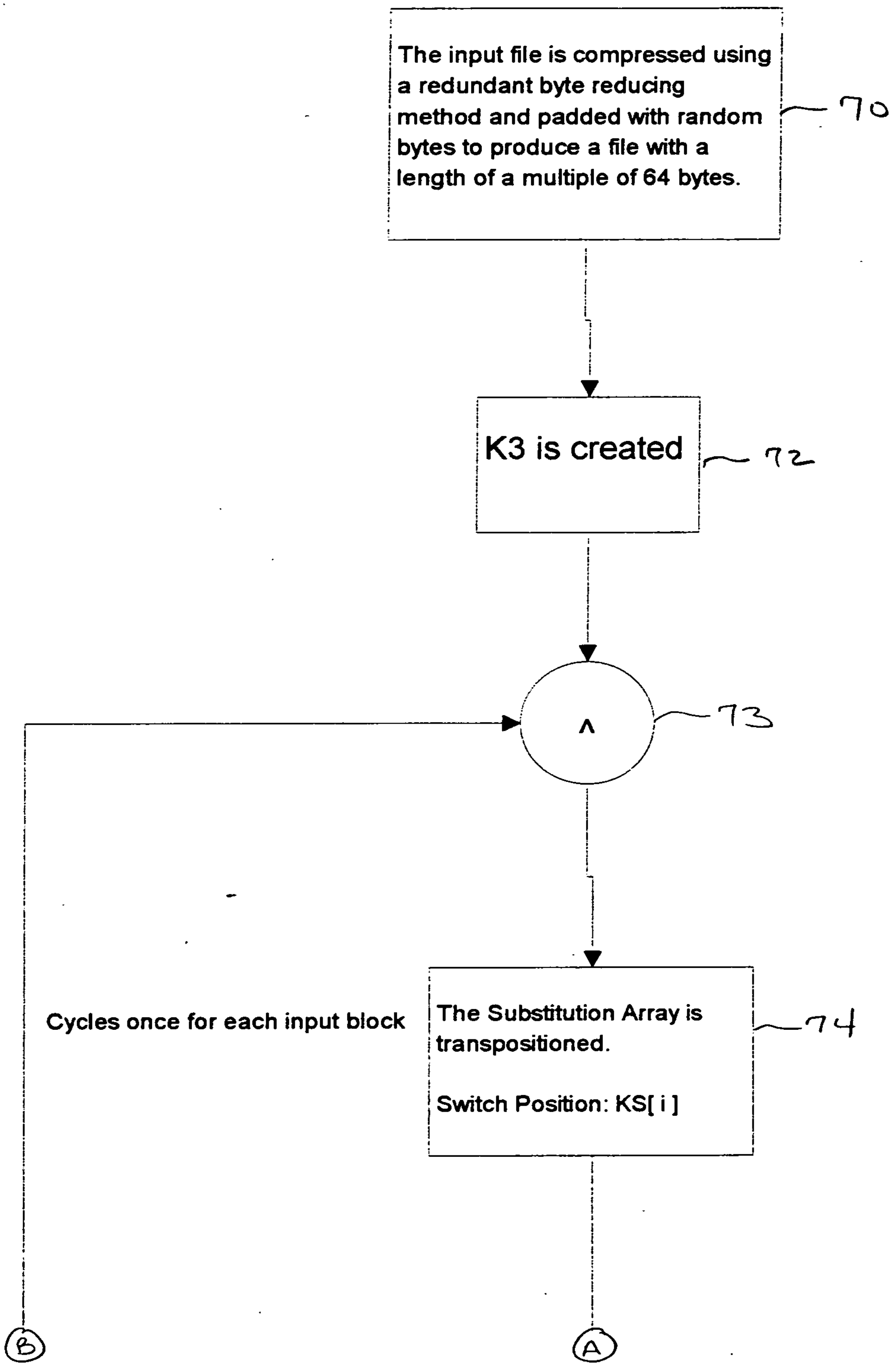

The input file is compressed using a redundant byte reducing method and padded with random bytes to produce a file with a length of a multiple of 64 bytes.
70
K3 is created
72
^
73
Cycles once for each input block
The Substitution Array is transpositioned.
Switch Position: KS[i]
74
B
A

FIG. 5

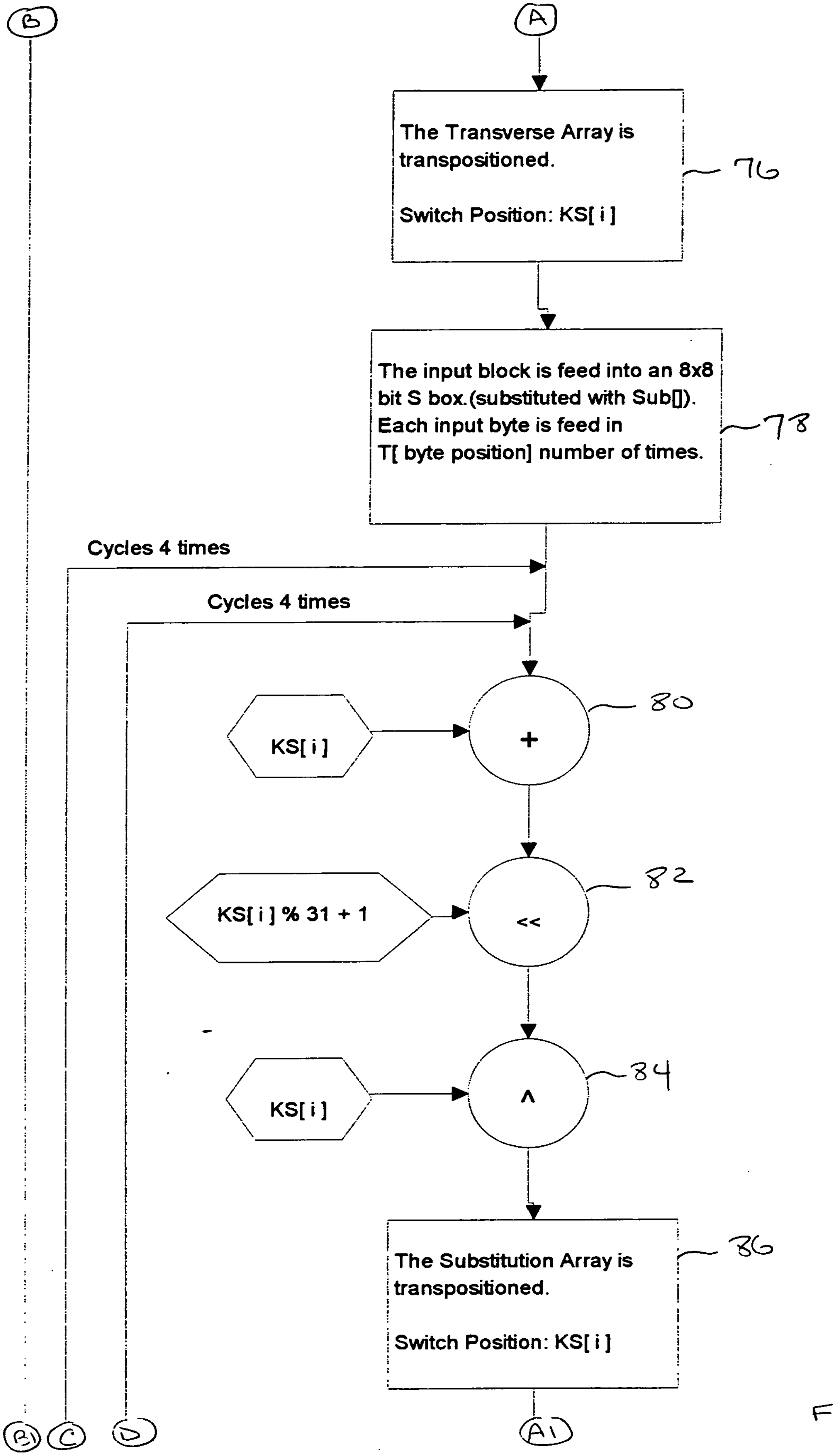

B
A
The Transverse Array is transpositioned.
Switch Position: KS[i]
76
The input block is feed into an 8x8 bit S box.(substituted with Sub[]). Each input byte is feed in T[byte position] number of times.
78
Cycles 4 times
Cycles 4 times
KS[i]
+
80
KS[i] % 31 + 1
<<
82
KS[i]
^
84
The Substitution Array is transpositioned.
Switch Position: KS[i]
86
B1
C
D
A1

FIG. 5 (CONT'D)

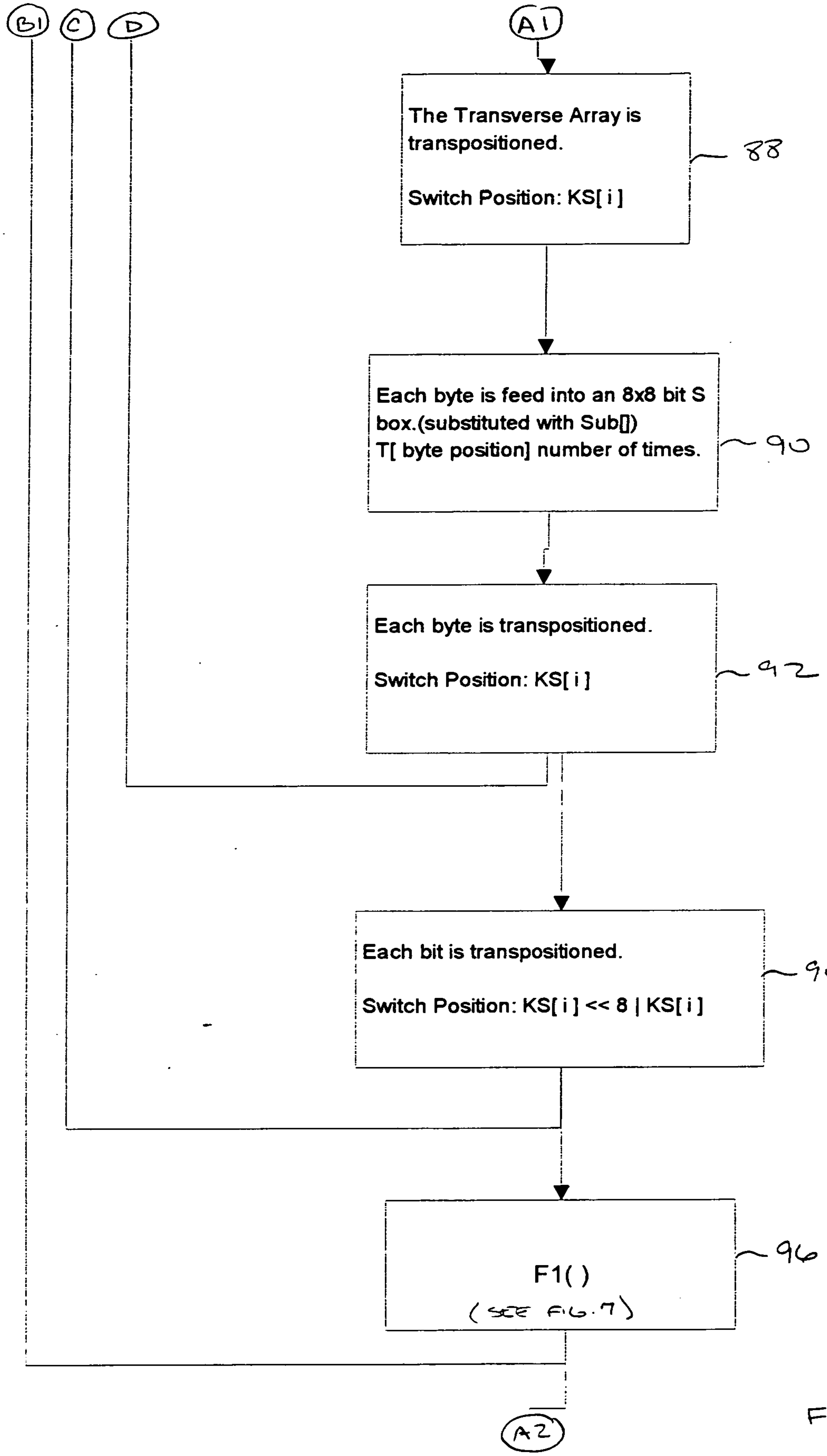
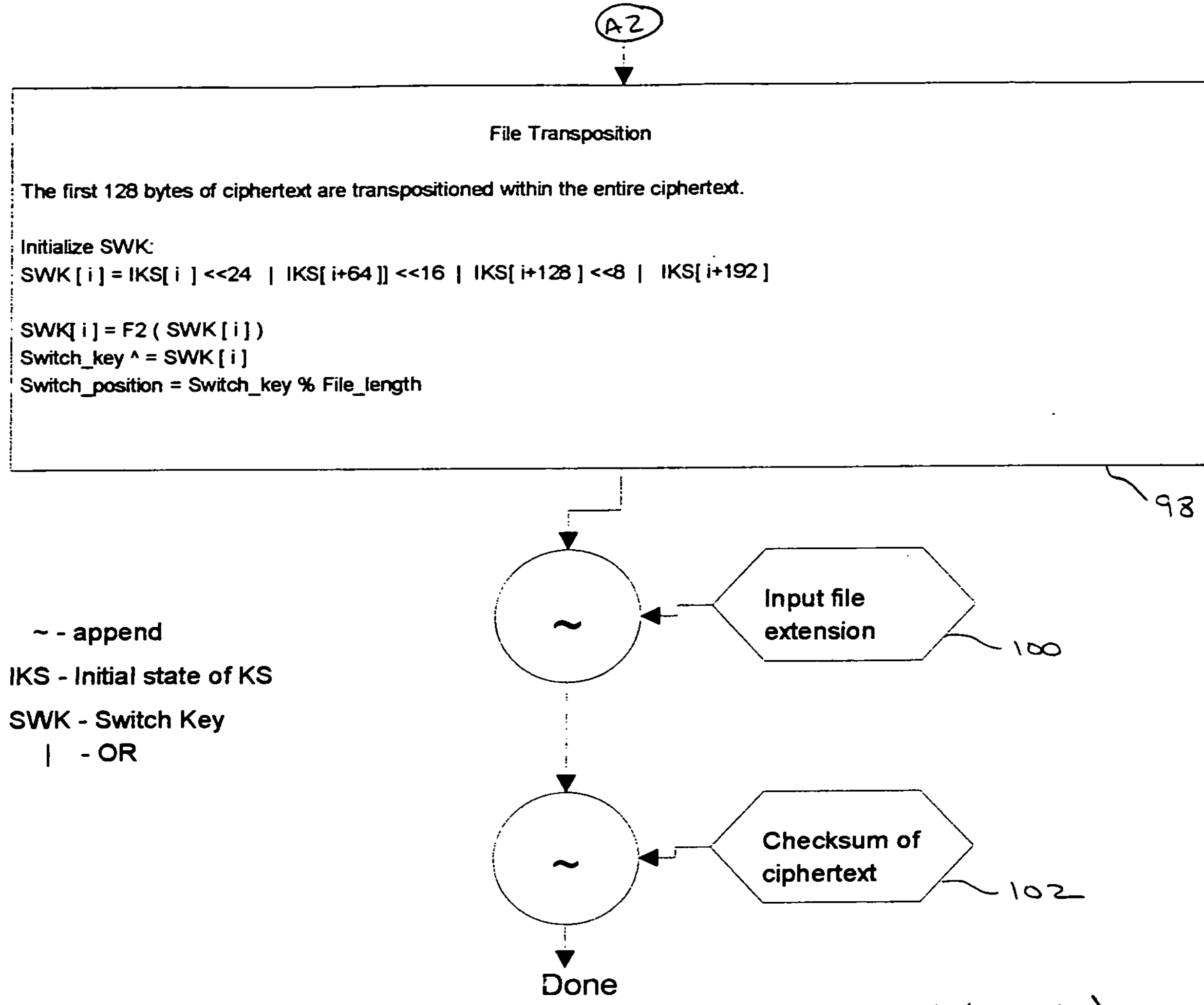

B1
C
D
A1
The Transverse Array is transpositioned.
Switch Position: KS[i]
88
Each byte is feed into an 8x8 bit S box.(substituted with Sub[])
T[byte position] number of times.
90
Each byte is transpositioned.
Switch Position: KS[i]
92
Each bit is transpositioned.
Switch Position: KS[i] << 8 | KS[i]
94
F1( )
(SEE FIG. 7)
96
A2

FIG. 5 (CONT'D)



File Transposition

The first 128 bytes of ciphertext are transpositioned within the entire ciphertext.

```
Initialize SWK:
SWK [ i ] = IKS[ i ] <<24  |  IKS[ i+64 ]] <<16  |  IKS[ i+128 ] <<8  |  IKS[ i+192 ]

SWK[ i ] = F2 ( SWK [ i ] )
Switch_key ^ = SWK [ i ]
Switch_position = Switch_key % File_length
```

FIG. 5 (CONT'D)

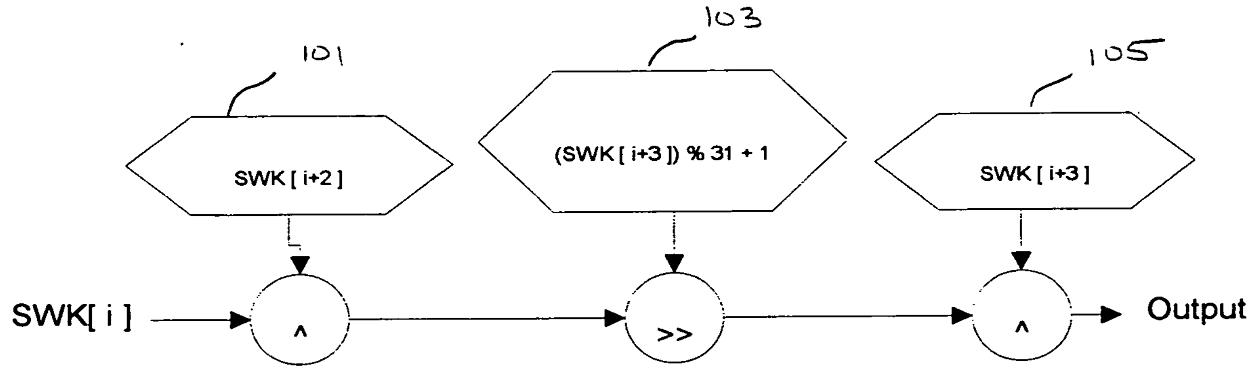


FIG. 6

K3 Modification

K3[i] + = (K2[K2[K3[i]]] % 255) + 113 + K2[i]

K1 Modification

K1_SEED ^ = K1[K1 [K3 [i]]]

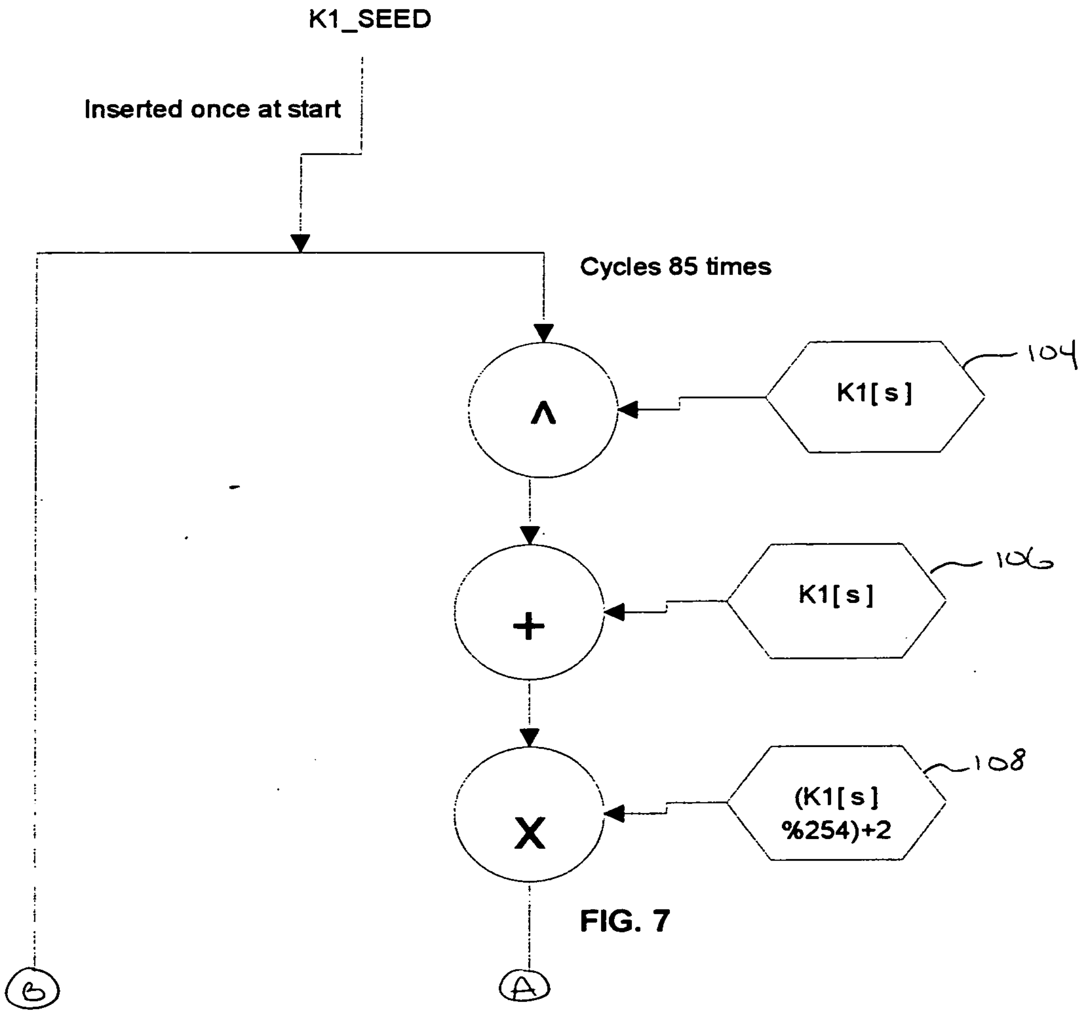


FIG. 7

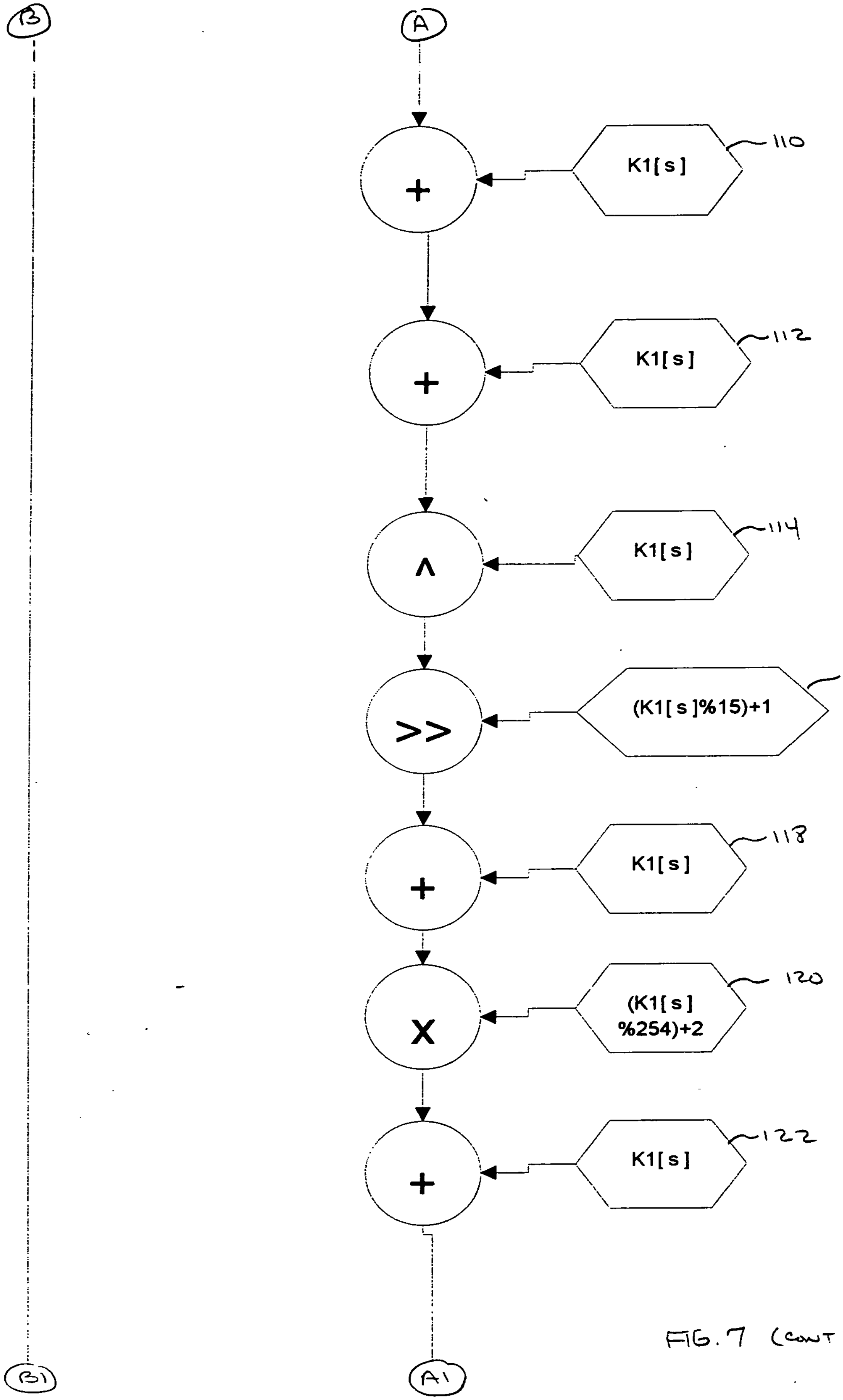

B
A
+
K1[s]
110
+
K1[s]
112
^
K1[s]
114
>>
(K1[s]%15)+1
116
+
K1[s]
118
X
(K1[s]
%254)+2
120
+
K1[s]
122
B1
A1

FIG. 7 (CONT'D)

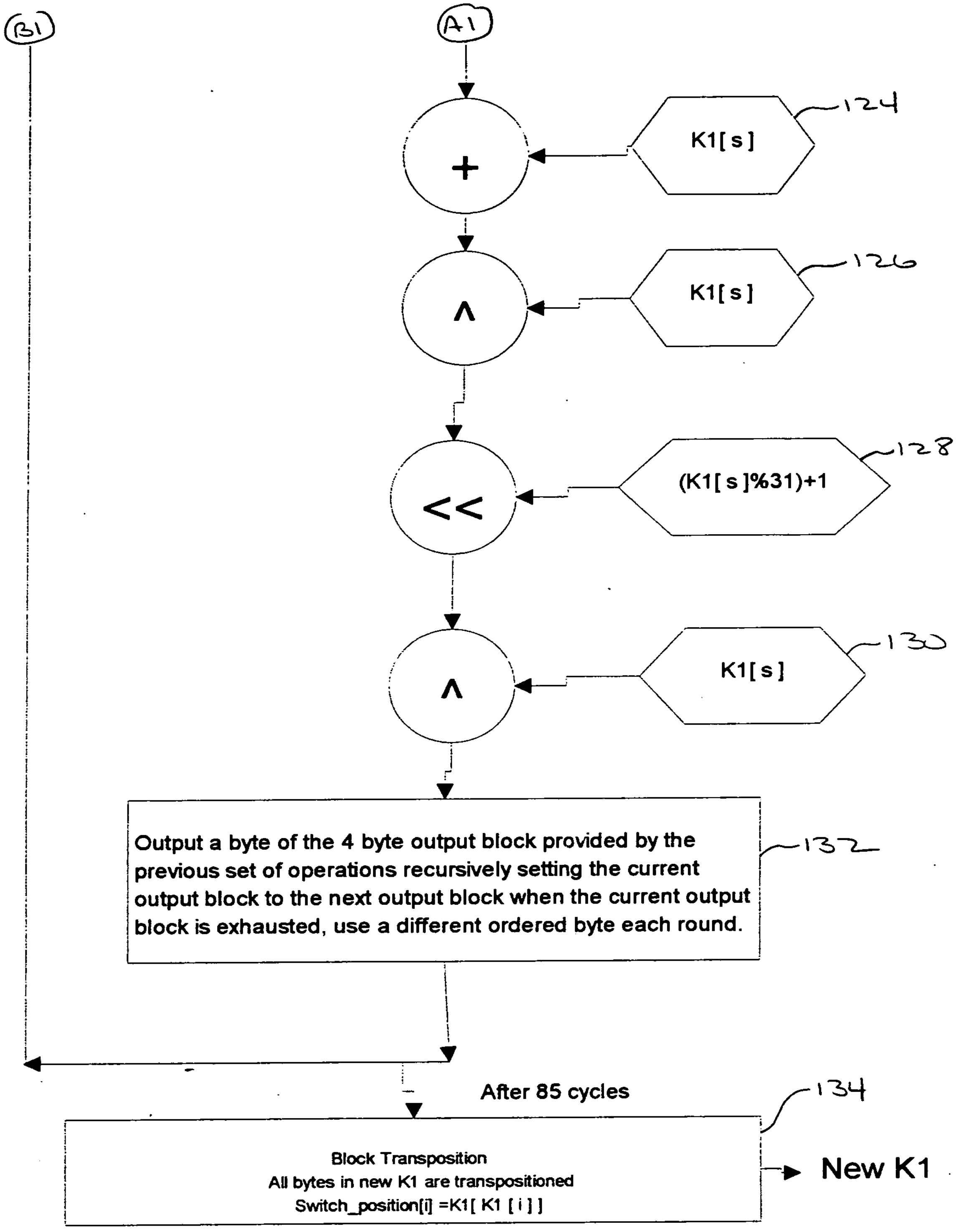

B1
A1
+
K1[s]
124
^
K1[s]
126
<<
(K1[s]%31)+1
128
^
K1[s]
130
Output a byte of the 4 byte output block provided by the previous set of operations recursively setting the current output block to the next output block when the current output block is exhausted, use a different ordered byte each round.
132
After 85 cycles
134
Block Transposition
All bytes in new K1 are transpositioned
Switch_position[i] =K1[K1 [i]]
New K1

FIG. 7 (CONT'D)

K2 Modification

K2_SEED + = (K3[K3 [#] % 64] % 253) + 3

K2_SEED ^ = K2 [K2 [K3 [K2[K3 [s % 64] + K2[#] % 192] % 64]]]

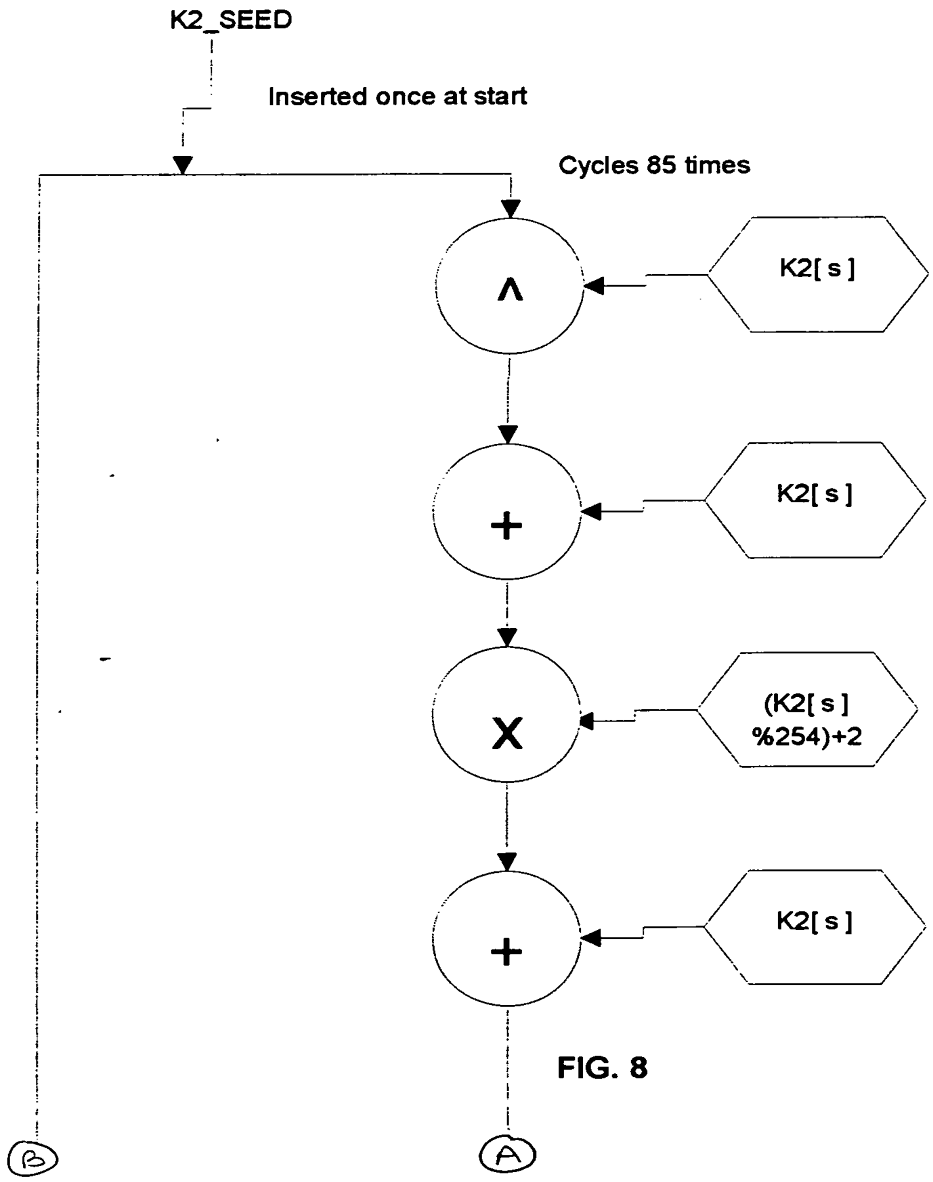


FIG. 8

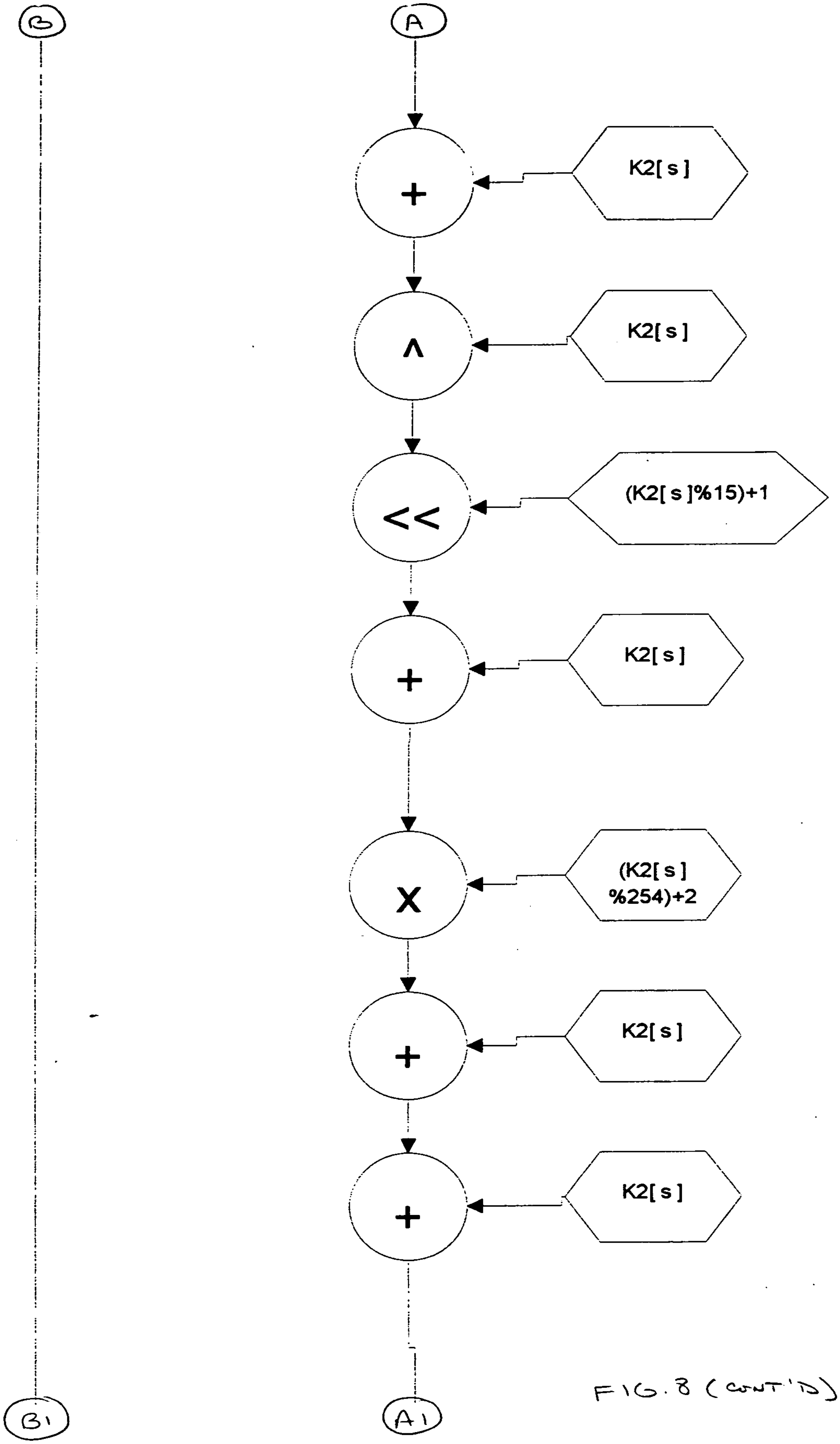


FIG. 8 (CONT'D)

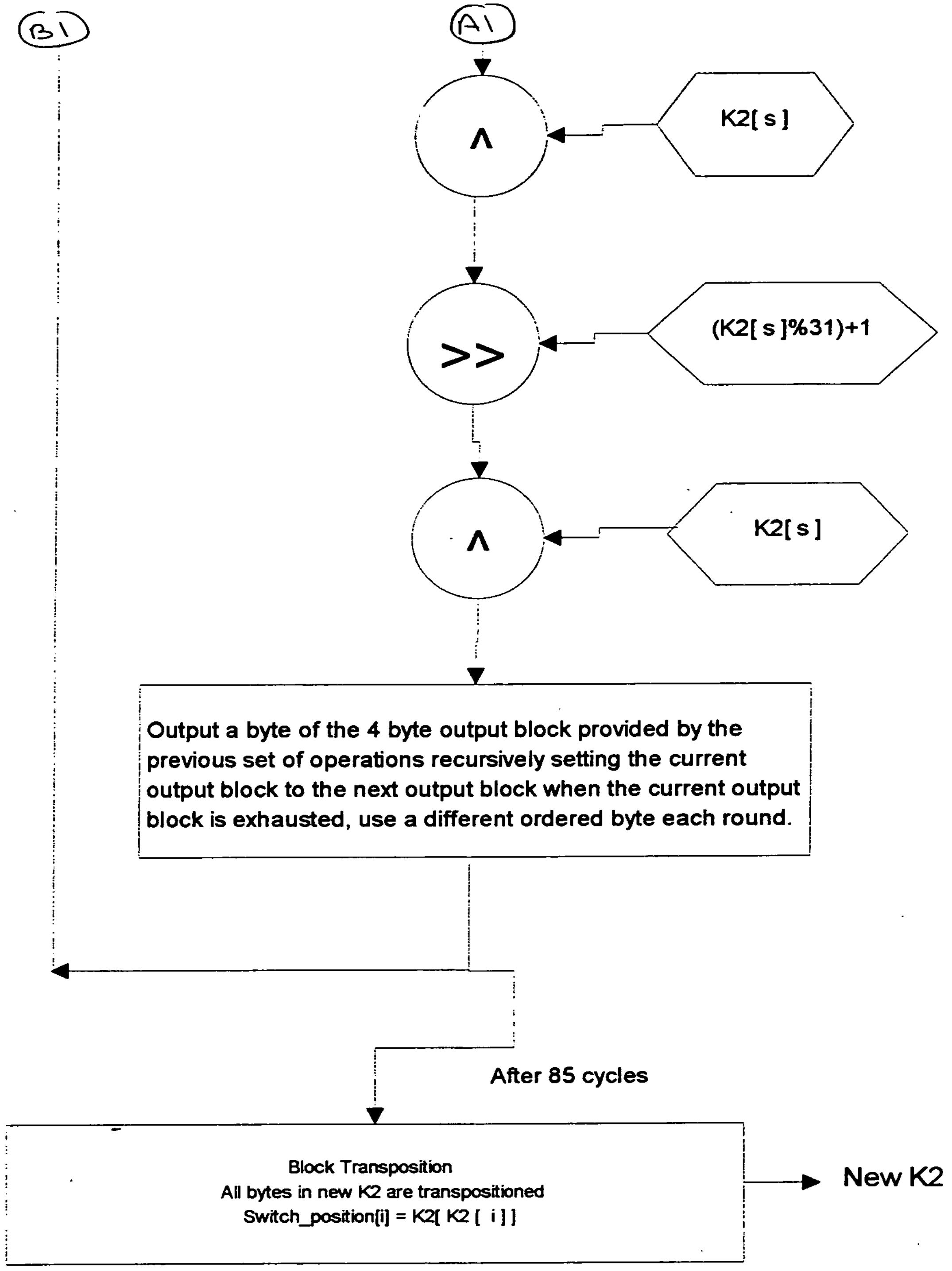

B1
A1
^
K2[s]
>>
(K2[s]%31)+1
^
K2[s]
Output a byte of the 4 byte output block provided by the previous set of operations recursively setting the current output block to the next output block when the current output block is exhausted, use a different ordered byte each round.
After 85 cycles
Block Transposition
All bytes in new K2 are transpositioned
Switch_position[i] = K2[K2 [i]]
New K2

FIG. 8 (CONT'D)

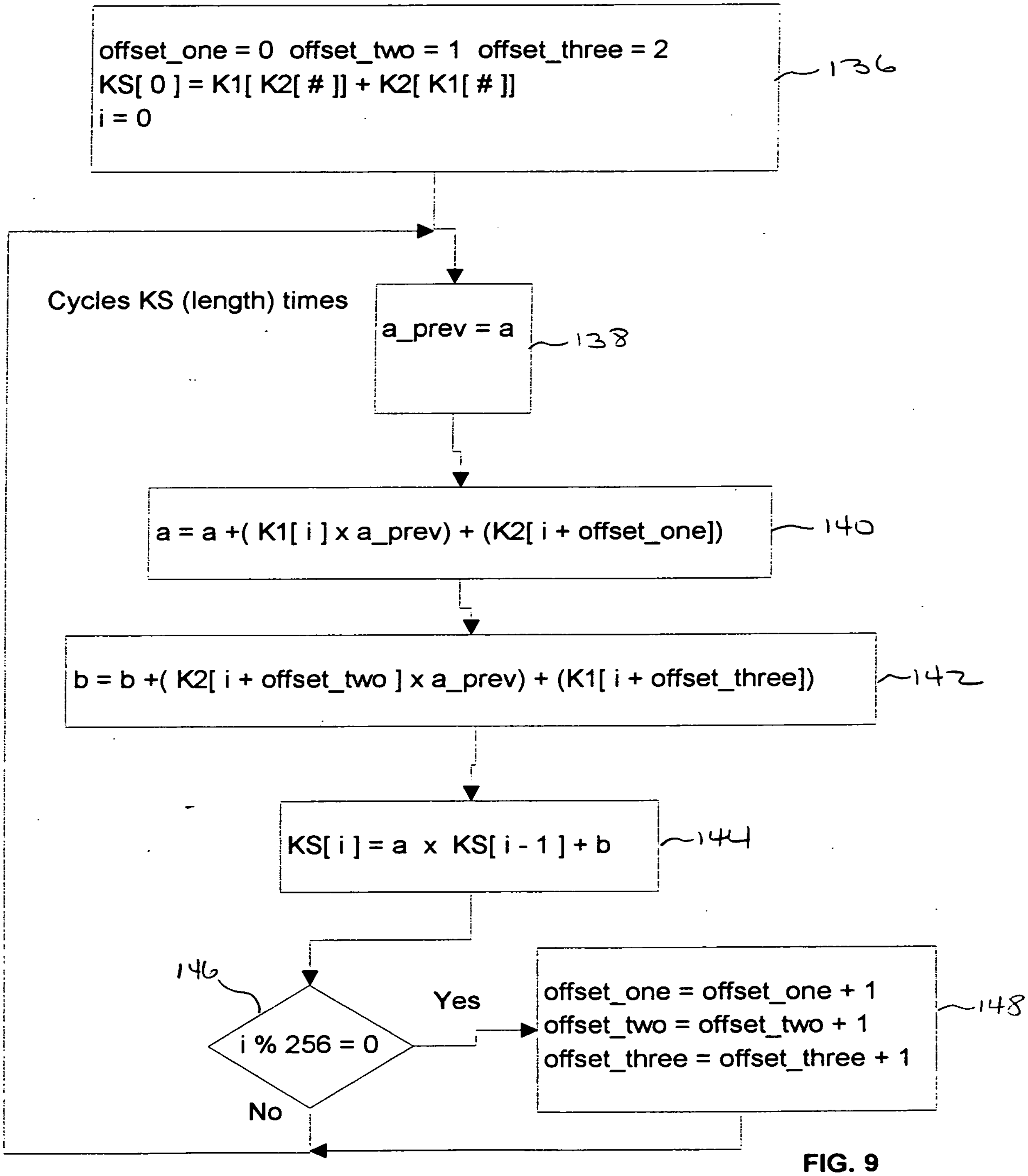

offset_one = 0 offset_two = 1 offset_three = 2
KS[0] = K1[K2[#]] + K2[K1[#]]
i = 0
136
Cycles KS (length) times
a_prev = a
138
a = a +(K1[i] x a_prev) + (K2[i + offset_one])
140
b = b +(K2[i + offset_two] x a_prev) + (K1[i + offset_three])
142
KS[i] = a x KS[i - 1] + b
144
146
i % 256 = 0
Yes
No
offset_one = offset_one + 1
offset_two = offset_two + 1
offset_three = offset_three + 1
148

FIG. 9

150

```
H1(v1,v2,v3,v4,v5,v6,v7) =  ( v1 ^  v2 &  v3 | ~v4 &  v5 ^  v6 ^  v7)
H2(v1,v2,v3,v4,v5,v6,v7)  = ( v1 & ~v2 ^  v3 ^  v4 ^  v5 &  v6 |  v7)
H3(v1,v2,v3,v4,v5,v6,v7)  = ( v1 ^  v2 |  v3 ^  v4 | ~v5 ^  v6 ^ ~v7)
H4(v1,v2,v3,v4,v5,v6,v7)  = (~v1 ^  v2 &  v3 |  v4 ^  v5 ^ ~v6 &  v7)
H5(v1,v2,v3,v4,v5,v6,v7)  = ( v1 &  v2 ^  v3 ^ ~v4 |  v5 &  v6 ^  v7)
H6(v1,v2,v3,v4,v5,v6,v7)  = ( v1 ^  v2 & ~v3 |  v4 &  v5 |  v6 ^  v7)
H7(v1,v2,v3,v4,v5,v6,v7)  = ( v1 ^  v2 |  v3 &  v4 ^  v5 ^ ~v6 &  v7)
H8(v1,v2,v3,v4,v5,v6,v7)  = (~v1 &  v2 ^  v3 |  v4 ^  v5 &  v6 ^  v7)

 HASH(hnum,output,v1,v2,v3,v4,v5,v6,v7,key) = (output +=
key+hnum(v1,v2,v3,v4,v5,v6,v7)

 HASH_FOR_KEY(hnum,result,output,v1,v2,v3,v4,v5,v6,v7,key) =
(result+=output+key+hnum(v1,v2,v3,v4,v5,v6,v7))
```

FIG. 10

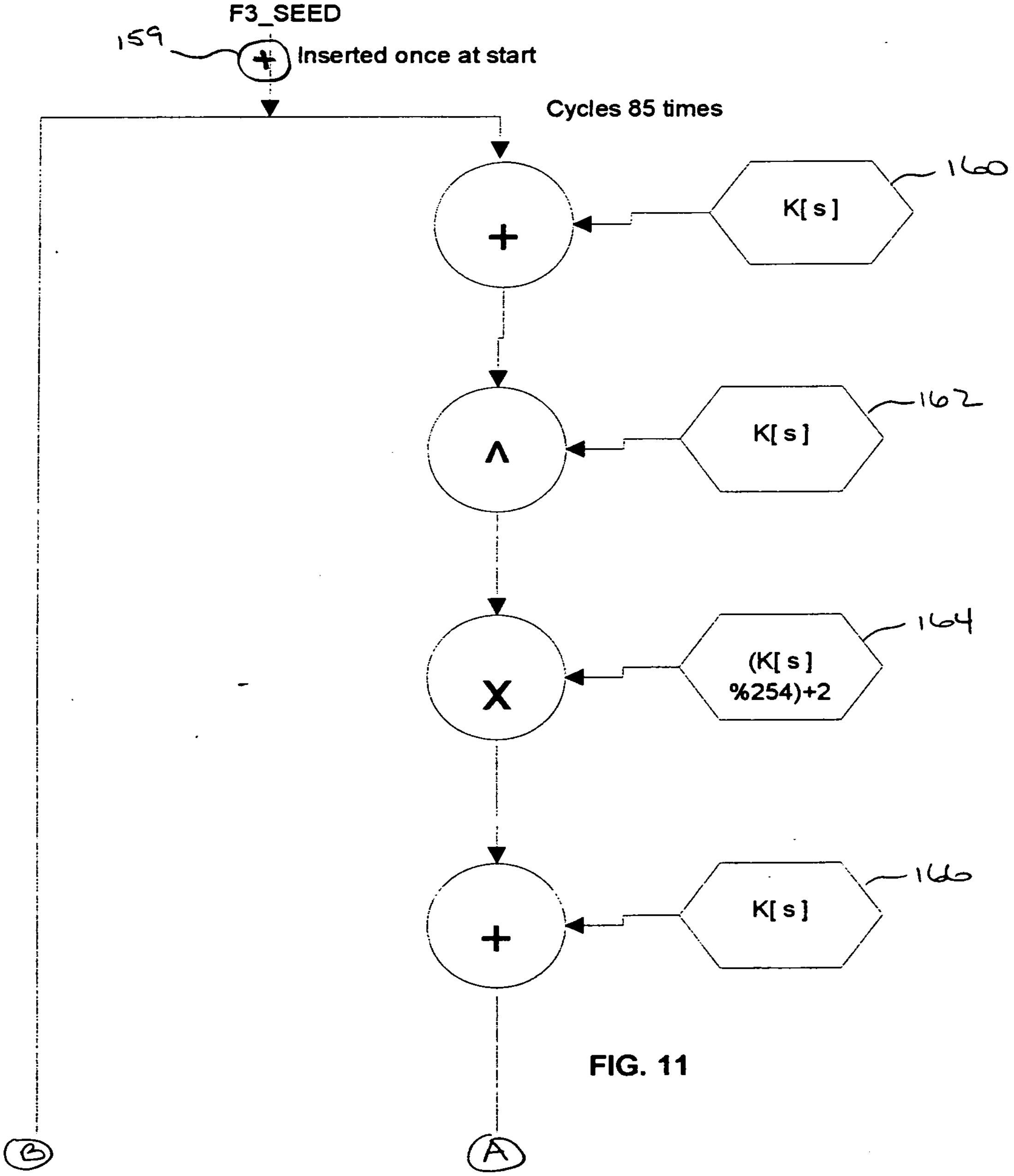

F3_SEED
159
+
Inserted once at start
Cycles 85 times
+
K[s]
160
^
K[s]
162
X
(K[s]
%254)+2
164
+
K[s]
166
B
A

FIG. 11

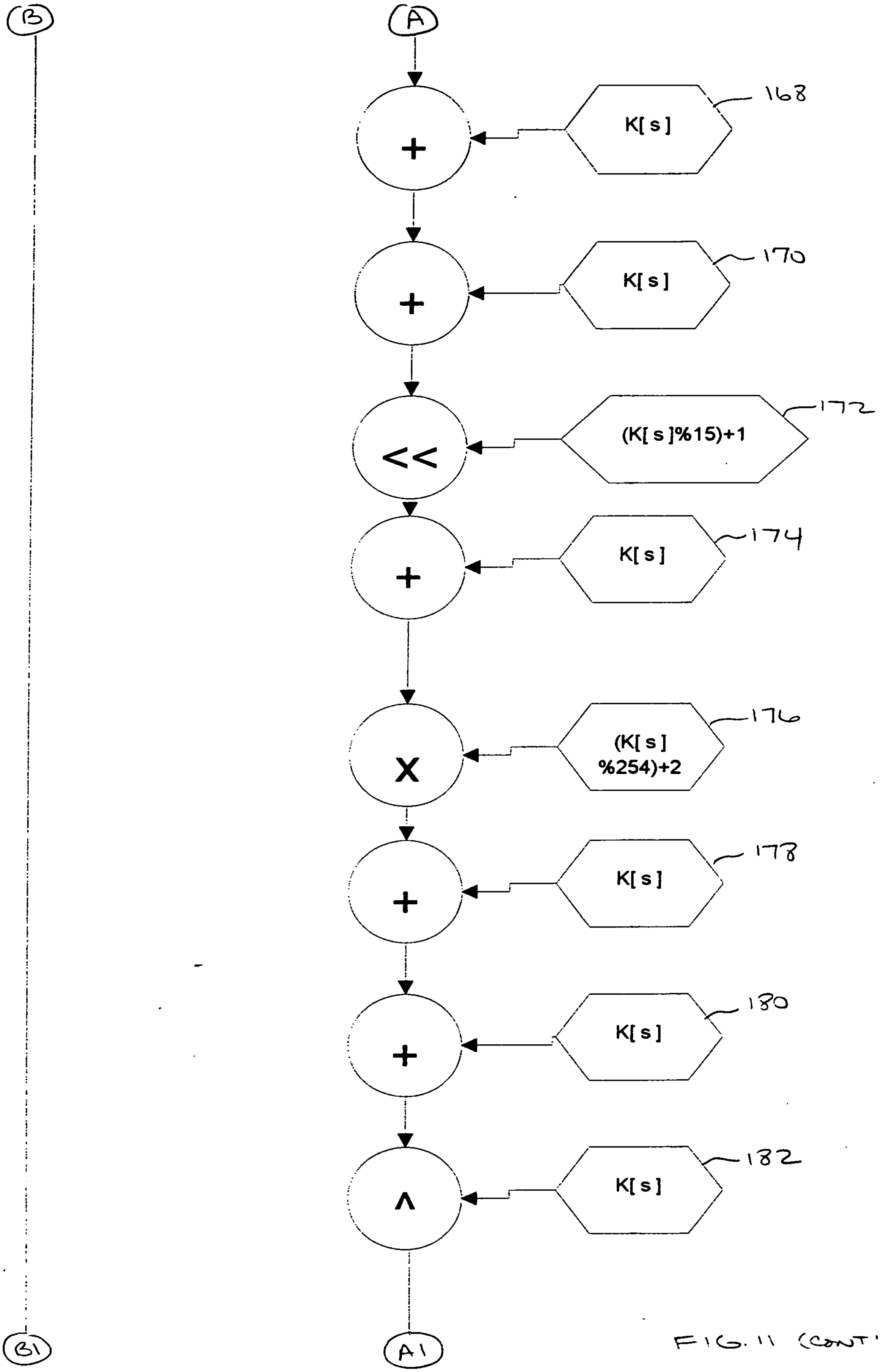
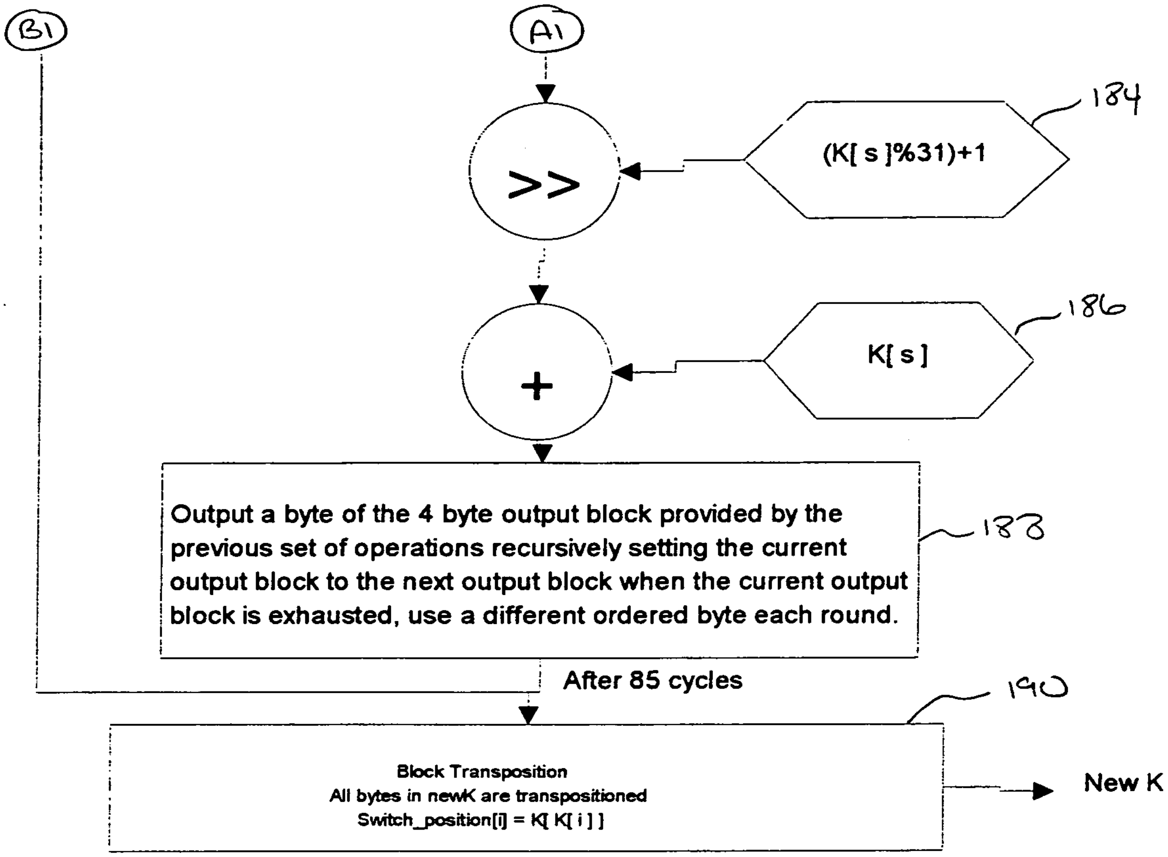

B
A
+
K[s]
168
+
K[s]
170
<<
(K[s]%15)+1
172
+
K[s]
174
X
(K[s]
%254)+2
176
+
K[s]
178
+
K[s]
180
^
K[s]
182
B1
A1

FIG. 11 (CONT'D)


B1
A1
>>
(K[s]%31)+1
184
+
K[s]
186
Output a byte of the 4 byte output block provided by the previous set of operations recursively setting the current output block to the next output block when the current output block is exhausted, use a different ordered byte each round.
188
After 85 cycles
190
Block Transposition
All bytes in newK are transpositioned
Switch_position[i] = K[K[i]]
New K

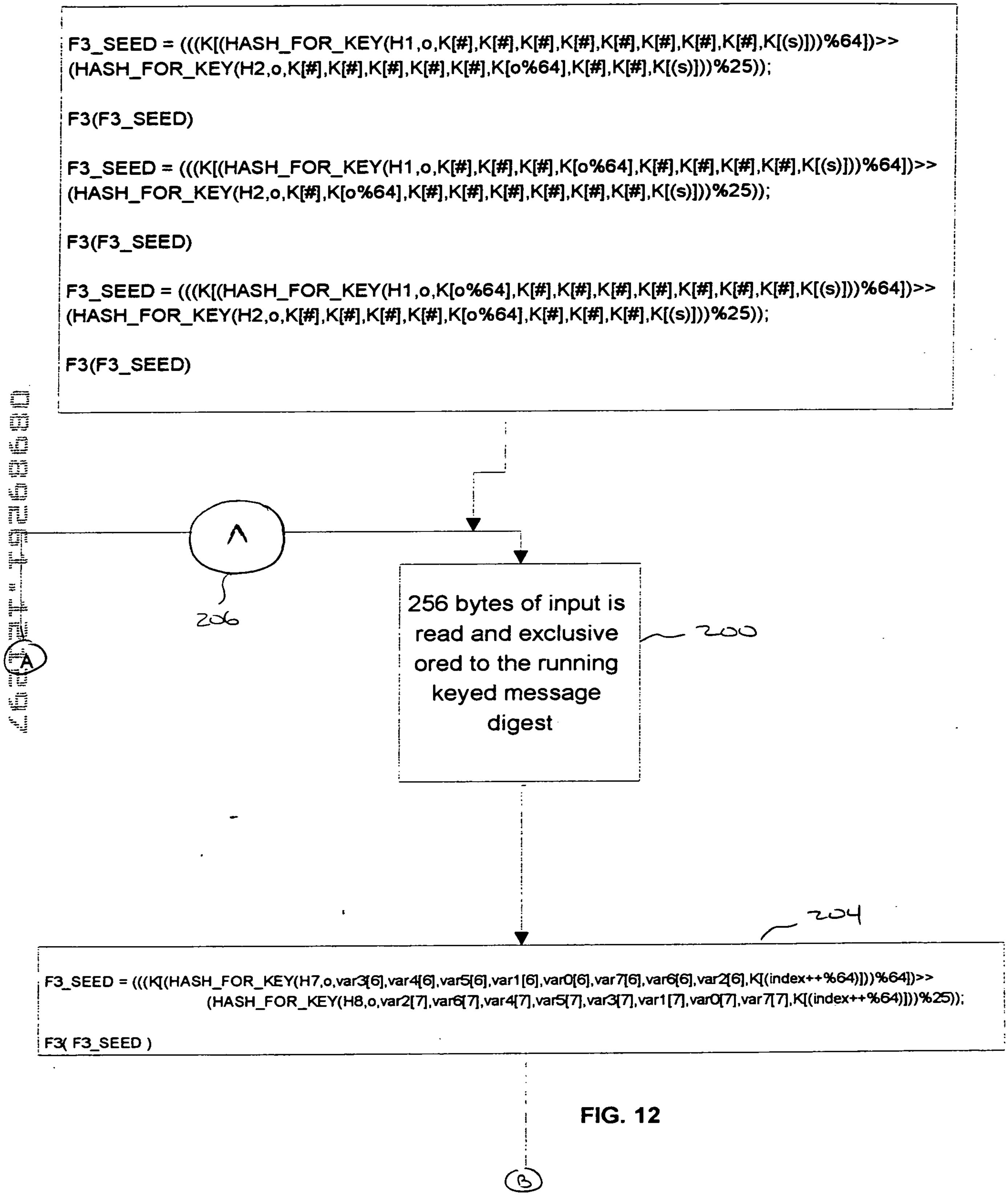
input_block = 256 bytes of input, read from the input file.

var0 = 32 bit pointer assigned to input_block;
var1 = 32 bit pointer assigned to (input_block+32);
var2 = 32 bit pointer assigned to (input_block+64);
var3 = 32 bit pointer assigned to (input_block+96);
var4 = 32 bit pointer assigned to (input_block+128);
var5 = 32 bit pointer assigned to (input_block+160);
var6 = 32 bit pointer assigned to (input_block+192);
var7 = 32 bit pointer assigned to (input_block+224);

- static numbers
index++ - running index
rep - running index

for(rep=0;rep<8;rep++){ } - Code within "{ }" will be executed eight times and rep will be incremented after each loop.

FIG. 11 (CONT'D)

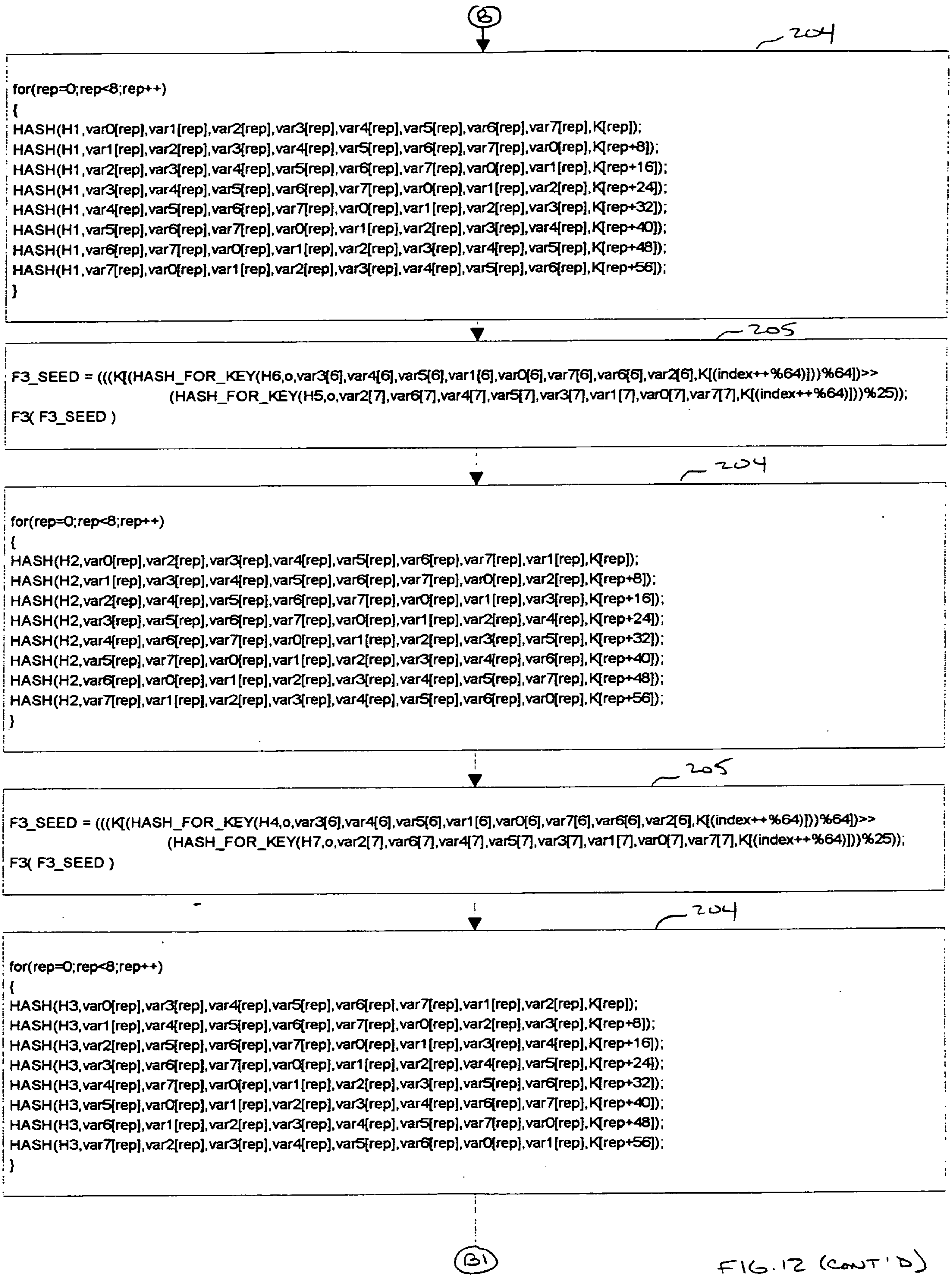


```
F3_SEED = (((K[(HASH_FOR_KEY(H1,o,K[#],K[#],K[#],K[#],K[#],K[#],K[#],K[#],K[(s)]))%64])>>
(HASH_FOR_KEY(H2,o,K[#],K[#],K[#],K[#],K[#],K[o%64],K[#],K[#],K[(s)]))%25));

F3(F3_SEED)

F3_SEED = (((K[(HASH_FOR_KEY(H1,o,K[#],K[#],K[#],K[o%64],K[#],K[#],K[#],K[#],K[(s)]))%64])>>
(HASH_FOR_KEY(H2,o,K[#],K[o%64],K[#],K[#],K[#],K[#],K[#],K[#],K[(s)]))%25));

F3(F3_SEED)

F3_SEED = (((K[(HASH_FOR_KEY(H1,o,K[o%64],K[#],K[#],K[#],K[#],K[#],K[#],K[#],K[(s)]))%64])>>
(HASH_FOR_KEY(H2,o,K[#],K[#],K[#],K[#],K[o%64],K[#],K[#],K[#],K[(s)]))%25));

F3(F3_SEED)
```

256 bytes of input is read and exclusive ored to the running keyed message digest

```
F3_SEED = (((K[(HASH_FOR_KEY(H7,o,var3[6],var4[6],var5[6],var1[6],var0[6],var7[6],var6[6],var2[6],K[(index++%64)]))%64])>>
                         (HASH_FOR_KEY(H8,o,var2[7],var6[7],var4[7],var5[7],var3[7],var1[7],var0[7],var7[7],K[(index++%64)]))%25));

F3( F3_SEED )
```

FIG. 12



```
for(rep=0;rep<8;rep++)
{
HASH(H1,var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],K[rep]);
HASH(H1,var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],K[rep+8]);
HASH(H1,var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],K[rep+16]);
HASH(H1,var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],K[rep+24]);
HASH(H1,var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],K[rep+32]);
HASH(H1,var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],K[rep+40]);
HASH(H1,var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],K[rep+48]);
HASH(H1,var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],K[rep+56]);
}
```

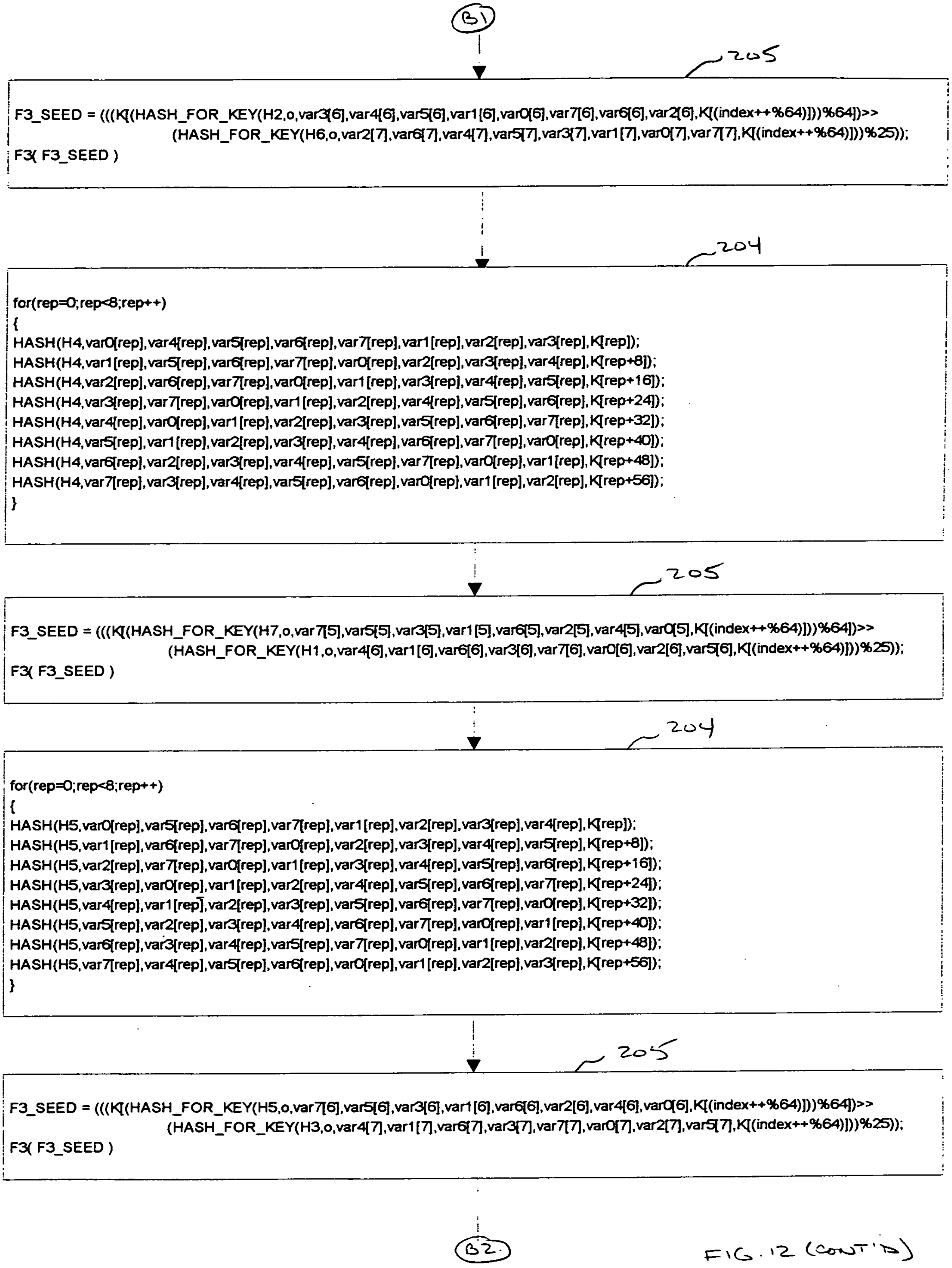
```
F3_SEED = (((K[(HASH_FOR_KEY(H6,o,var3[6],var4[6],var5[6],var1[6],var0[6],var7[6],var6[6],var2[6],K[(index++%64)]))%64])>>
                        (HASH_FOR_KEY(H5,o,var2[7],var6[7],var4[7],var5[7],var3[7],var1[7],var0[7],var7[7],K[(index++%64)]))%25));
F3( F3_SEED )
```

```
for(rep=0;rep<8;rep++)
{
HASH(H2,var0[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],var1[rep],K[rep]);
HASH(H2,var1[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],var2[rep],K[rep+8]);
HASH(H2,var2[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],var3[rep],K[rep+16]);
HASH(H2,var3[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],var4[rep],K[rep+24]);
HASH(H2,var4[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],var5[rep],K[rep+32]);
HASH(H2,var5[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],var6[rep],K[rep+40]);
HASH(H2,var6[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],var7[rep],K[rep+48]);
HASH(H2,var7[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],var0[rep],K[rep+56]);
}
```

```
F3_SEED = (((K[(HASH_FOR_KEY(H4,o,var3[6],var4[6],var5[6],var1[6],var0[6],var7[6],var6[6],var2[6],K[(index++%64)]))%64])>>
                        (HASH_FOR_KEY(H7,o,var2[7],var6[7],var4[7],var5[7],var3[7],var1[7],var0[7],var7[7],K[(index++%64)]))%25));
F3( F3_SEED )
```

```
for(rep=0;rep<8;rep++)
{
HASH(H3,var0[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],var1[rep],var2[rep],K[rep]);
HASH(H3,var1[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],var2[rep],var3[rep],K[rep+8]);
HASH(H3,var2[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],var3[rep],var4[rep],K[rep+16]);
HASH(H3,var3[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],var4[rep],var5[rep],K[rep+24]);
HASH(H3,var4[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],var5[rep],var6[rep],K[rep+32]);
HASH(H3,var5[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],var6[rep],var7[rep],K[rep+40]);
HASH(H3,var6[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],var7[rep],var0[rep],K[rep+48]);
HASH(H3,var7[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],var0[rep],var1[rep],K[rep+56]);
}
```

FIG. 12 (CONT'D)



```
F3_SEED = (((K[(HASH_FOR_KEY(H2,o,var3[6],var4[6],var5[6],var1[6],var0[6],var7[6],var6[6],var2[6],K[(index++%64)]))%64])>>
                        (HASH_FOR_KEY(H6,o,var2[7],var6[7],var4[7],var5[7],var3[7],var1[7],var0[7],var7[7],K[(index++%64)]))%25));
F3( F3_SEED )
```

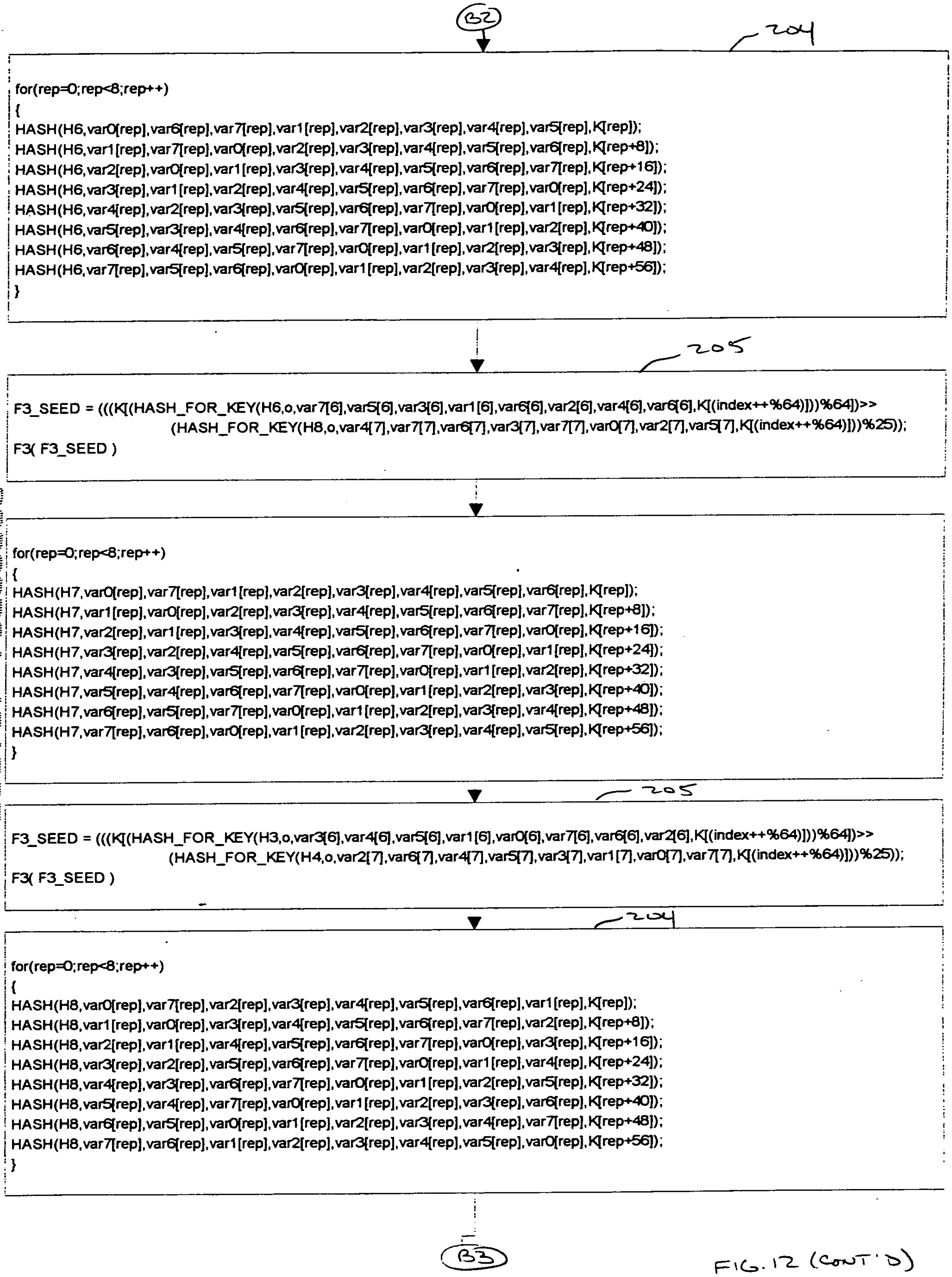
```
for(rep=0;rep<8;rep++)
{
HASH(H4,var0[rep],var4[rep],var5[rep],var6[rep],var7[rep],var1[rep],var2[rep],var3[rep],K[rep]);
HASH(H4,var1[rep],var5[rep],var6[rep],var7[rep],var0[rep],var2[rep],var3[rep],var4[rep],K[rep+8]);
HASH(H4,var2[rep],var6[rep],var7[rep],var0[rep],var1[rep],var3[rep],var4[rep],var5[rep],K[rep+16]);
HASH(H4,var3[rep],var7[rep],var0[rep],var1[rep],var2[rep],var4[rep],var5[rep],var6[rep],K[rep+24]);
HASH(H4,var4[rep],var0[rep],var1[rep],var2[rep],var3[rep],var5[rep],var6[rep],var7[rep],K[rep+32]);
HASH(H4,var5[rep],var1[rep],var2[rep],var3[rep],var4[rep],var6[rep],var7[rep],var0[rep],K[rep+40]);
HASH(H4,var6[rep],var2[rep],var3[rep],var4[rep],var5[rep],var7[rep],var0[rep],var1[rep],K[rep+48]);
HASH(H4,var7[rep],var3[rep],var4[rep],var5[rep],var6[rep],var0[rep],var1[rep],var2[rep],K[rep+56]);
}
```

```
F3_SEED = (((K[(HASH_FOR_KEY(H7,o,var7[5],var5[5],var3[5],var1[5],var6[5],var2[5],var4[5],var0[5],K[(index++%64)]))%64])>>
                        (HASH_FOR_KEY(H1,o,var4[6],var1[6],var6[6],var3[6],var7[6],var0[6],var2[6],var5[6],K[(index++%64)]))%25));
F3( F3_SEED )
```

```
for(rep=0;rep<8;rep++)
{
HASH(H5,var0[rep],var5[rep],var6[rep],var7[rep],var1[rep],var2[rep],var3[rep],var4[rep],K[rep]);
HASH(H5,var1[rep],var6[rep],var7[rep],var0[rep],var2[rep],var3[rep],var4[rep],var5[rep],K[rep+8]);
HASH(H5,var2[rep],var7[rep],var0[rep],var1[rep],var3[rep],var4[rep],var5[rep],var6[rep],K[rep+16]);
HASH(H5,var3[rep],var0[rep],var1[rep],var2[rep],var4[rep],var5[rep],var6[rep],var7[rep],K[rep+24]);
HASH(H5,var4[rep],var1[rep],var2[rep],var3[rep],var5[rep],var6[rep],var7[rep],var0[rep],K[rep+32]);
HASH(H5,var5[rep],var2[rep],var3[rep],var4[rep],var6[rep],var7[rep],var0[rep],var1[rep],K[rep+40]);
HASH(H5,var6[rep],var3[rep],var4[rep],var5[rep],var7[rep],var0[rep],var1[rep],var2[rep],K[rep+48]);
HASH(H5,var7[rep],var4[rep],var5[rep],var6[rep],var0[rep],var1[rep],var2[rep],var3[rep],K[rep+56]);
}
```

```
F3_SEED = (((K[(HASH_FOR_KEY(H5,o,var7[6],var5[6],var3[6],var1[6],var6[6],var2[6],var4[6],var0[6],K[(index++%64)]))%64])>>
                        (HASH_FOR_KEY(H3,o,var4[7],var1[7],var6[7],var3[7],var7[7],var0[7],var2[7],var5[7],K[(index++%64)]))%25));
F3( F3_SEED )
```

FIG. 12 (CONT'D)



```
for(rep=0;rep<8;rep++)
{
HASH(H6,var0[rep],var6[rep],var7[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],K[rep]);
HASH(H6,var1[rep],var7[rep],var0[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],K[rep+8]);
HASH(H6,var2[rep],var0[rep],var1[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],K[rep+16]);
HASH(H6,var3[rep],var1[rep],var2[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],K[rep+24]);
HASH(H6,var4[rep],var2[rep],var3[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],K[rep+32]);
HASH(H6,var5[rep],var3[rep],var4[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],K[rep+40]);
HASH(H6,var6[rep],var4[rep],var5[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],K[rep+48]);
HASH(H6,var7[rep],var5[rep],var6[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],K[rep+56]);
}
```

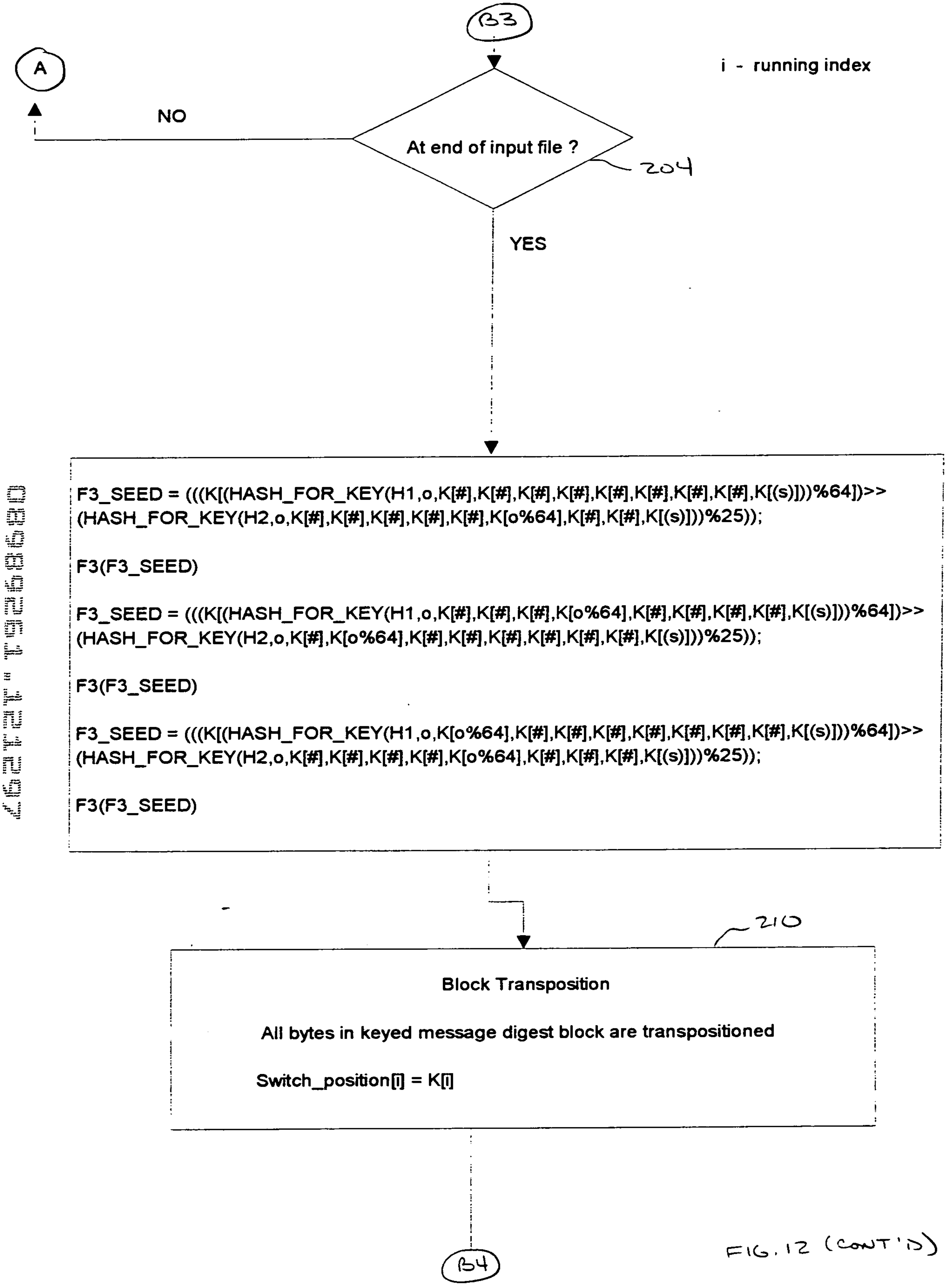
```
F3_SEED = (((K[(HASH_FOR_KEY(H6,o,var7[6],var5[6],var3[6],var1[6],var6[6],var2[6],var4[6],var6[6],K[(index++%64)]))%64])>>
                    (HASH_FOR_KEY(H8,o,var4[7],var7[7],var6[7],var3[7],var7[7],var0[7],var2[7],var5[7],K[(index++%64)]))%25));
F3( F3_SEED )
```

```
for(rep=0;rep<8;rep++)
{
HASH(H7,var0[rep],var7[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],K[rep]);
HASH(H7,var1[rep],var0[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],K[rep+8]);
HASH(H7,var2[rep],var1[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],K[rep+16]);
HASH(H7,var3[rep],var2[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],K[rep+24]);
HASH(H7,var4[rep],var3[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],K[rep+32]);
HASH(H7,var5[rep],var4[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],K[rep+40]);
HASH(H7,var6[rep],var5[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],K[rep+48]);
HASH(H7,var7[rep],var6[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],K[rep+56]);
}
```

```
F3_SEED = (((K[(HASH_FOR_KEY(H3,o,var3[6],var4[6],var5[6],var1[6],var0[6],var7[6],var6[6],var2[6],K[(index++%64)]))%64])>>
                    (HASH_FOR_KEY(H4,o,var2[7],var6[7],var4[7],var5[7],var3[7],var1[7],var0[7],var7[7],K[(index++%64)]))%25));
F3( F3_SEED )
```

```
for(rep=0;rep<8;rep++)
{
HASH(H8,var0[rep],var7[rep],var2[rep],var3[rep],var4[rep],var5[rep],var6[rep],var1[rep],K[rep]);
HASH(H8,var1[rep],var0[rep],var3[rep],var4[rep],var5[rep],var6[rep],var7[rep],var2[rep],K[rep+8]);
HASH(H8,var2[rep],var1[rep],var4[rep],var5[rep],var6[rep],var7[rep],var0[rep],var3[rep],K[rep+16]);
HASH(H8,var3[rep],var2[rep],var5[rep],var6[rep],var7[rep],var0[rep],var1[rep],var4[rep],K[rep+24]);
HASH(H8,var4[rep],var3[rep],var6[rep],var7[rep],var0[rep],var1[rep],var2[rep],var5[rep],K[rep+32]);
HASH(H8,var5[rep],var4[rep],var7[rep],var0[rep],var1[rep],var2[rep],var3[rep],var6[rep],K[rep+40]);
HASH(H8,var6[rep],var5[rep],var0[rep],var1[rep],var2[rep],var3[rep],var4[rep],var7[rep],K[rep+48]);
HASH(H8,var7[rep],var6[rep],var1[rep],var2[rep],var3[rep],var4[rep],var5[rep],var0[rep],K[rep+56]);
}
```

FIG. 12 (CONT'D)



```
F3_SEED = (((K[(HASH_FOR_KEY(H1,o,K[#],K[#],K[#],K[#],K[#],K[#],K[#],K[#],K[(s)]))%64])>>
(HASH_FOR_KEY(H2,o,K[#],K[#],K[#],K[#],K[#],K[o%64],K[#],K[#],K[(s)]))%25));

F3(F3_SEED)

F3_SEED = (((K[(HASH_FOR_KEY(H1,o,K[#],K[#],K[#],K[o%64],K[#],K[#],K[#],K[#],K[(s)]))%64])>>
(HASH_FOR_KEY(H2,o,K[#],K[o%64],K[#],K[#],K[#],K[#],K[#],K[#],K[(s)]))%25));

F3(F3_SEED)

F3_SEED = (((K[(HASH_FOR_KEY(H1,o,K[o%64],K[#],K[#],K[#],K[#],K[#],K[#],K[#],K[(s)]))%64])>>
(HASH_FOR_KEY(H2,o,K[#],K[#],K[#],K[#],K[o%64],K[#],K[#],K[#],K[(s)]))%25));

F3(F3_SEED)
```

All bytes in keyed message digest block are transpositioned

Switch_position[i] = K[i]

FIG. 12 (CONT'D)

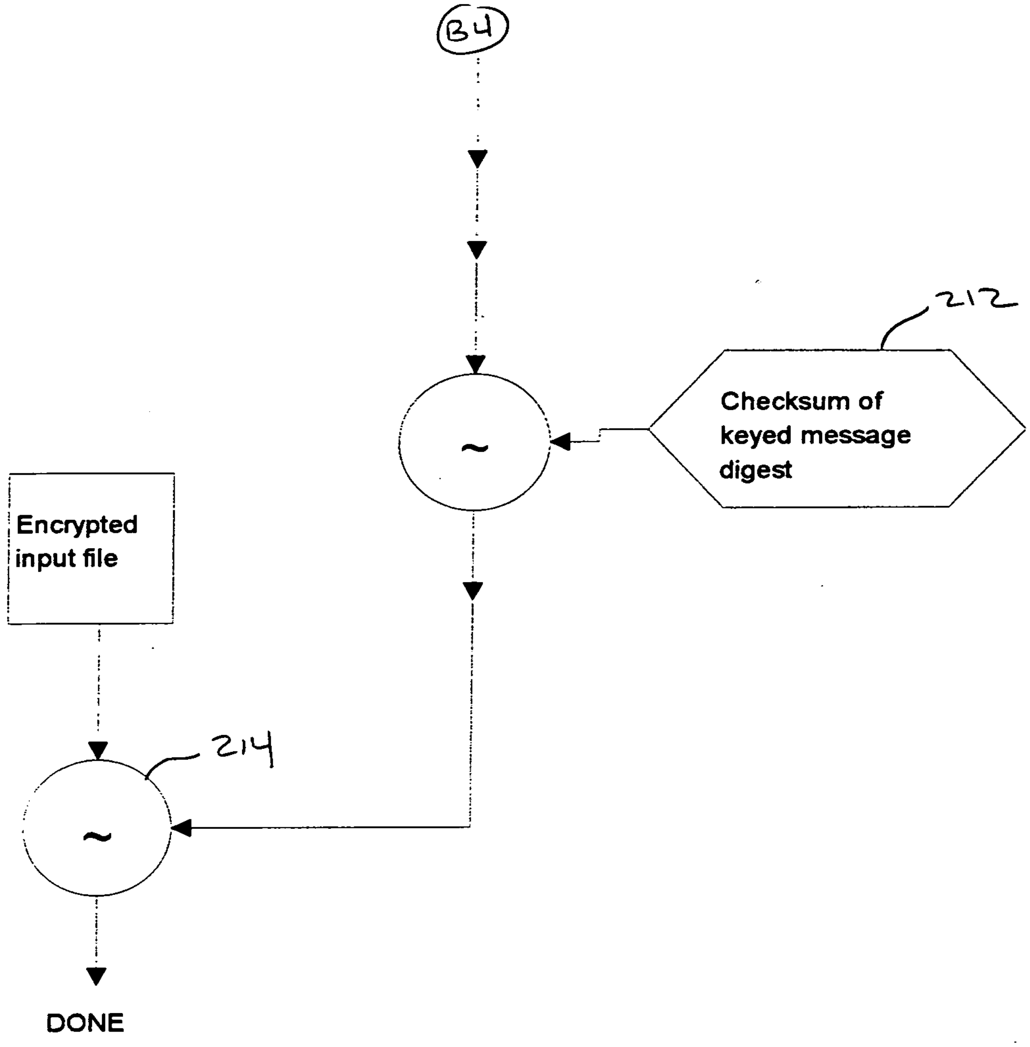

B4
212
~
Checksum of keyed message digest
Encrypted input file
214
~
DONE

FIG. 12 (CONT'D)